```
/*
UTMconsersion.h - Library to convert in UTM coordenates.
TinyTrackGPS v0.4

Copyright © 2019-2021 Francisco Rafael Reyes Carmona.
All rights reserved.

rafael.reyes.carmona@gmail.com

  This file is part of TinyTrackGPS.

  TinyTrackGPS is free software: you can redistribute it and/or
  modify
  it under the terms of the GNU General Public License as published
  by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  TinyTrackGPS is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with TinyTrackGPS.  If not, see <https://www.gnu.org/
  licenses/>.
*/

#if ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

#ifndef UTMconversion_h
#define UTMconversion_h

class GPS_UTM {
  private:
    int _h;
    char _letter;
    long _x;
    long _y;

  public:
    GPS_UTM(){
      _h = 0;
      _letter = 'Z';
```

```
    _letter = 'Z';
    _x = 0L;
    _y = 0L;
  };

  void UTM(double lati, double longi) {
    /*!
    * Transformación de las coordenadas geográficas a UTM
    */
    /* // Sobre la geometría del delipsoide WGS84
    double a = 6378137.0;
    double b = 6356752.3142;

    //  float e = sqrt((a*a) - (b*b))/a; ///< Excentricidad.
    double e = sqrt(sq(a) - sq(b))/b; ///< Segunda excentricidad.
    double e2 = sq(e); ///< al cuadrado. Usaremos esta
    directamente.
    double c = sq(a) / b; ///< Radio Polar de Curvatura.
    */

    /// Se realiza las declaraciones para agilizar el calculo a
    UTM de X e Y.
    double e2 = 673949675659e-14; ///< Segunda excentricidad al
    cuadrado.
    double e2_2 = 336974837829e-14; // (e2 / 2.0)
    double c = 639959362580397e-8; ///< Radio Polar de Curvatura.
    double PI_180 = 1745329251994e-14;// (PI / 180.0)
    double alf = 505462256744e-14; // (0.75 * e2)
    double bet = 425820155e-13; // ((5.0 / 3.0) * alf * alf)
    double gam = 16740579e-14; // ((35.0 / 27.0) * alf * alf *
    alf)

    /// Sobre la longitud y latitud. Conversión de grados
    decimales a radianes.

    /*!
    * Cálculo del signo de la longitud:
    *      - Si la longitud está referida al Oeste del meridiano
    de Greenwich,
    *        entonces la longitud es negativa (-).
    *      - Si la longitud está referida al Este del meridiano
    de Greenwich,
    *        entonces la longitud es positiva (+).
    */

    double latRad = lati * PI_180; ///< Latitud en Radianes.
    double lonRad = longi * PI_180; ///< Longitud en Radianes.

```

```
86          /// Sobre el huso.
87          float huso = ((longi + 180.0) / 6.0) + 1.0;  ///< Nos
            interesa quedarnos solo con la parte entera.
88          _h = (int)huso;
89
90          // Handle special case of west coast of Norway
91          if ( lati >= 56.0 && lati < 64.0 && longi >= 3.0 && longi <
            12.0 ) {
92            _h = 32;
93          }
94
95          // Special zones for Svalbard
96          if ( lati >= 72.0 && lati < 84.0 ) {
97            if ( longi >= 0.0  && longi <  9.0 ) {
98              _h = 31;
99            }
100           else if ( longi >= 9.0  && longi < 21.0 ) {
101             _h = 33;
102           }
103           else if ( longi >= 21.0 && longi < 33.0 ) {
104             _h = 35;
105           }
106           else if ( longi >= 33.0 && longi < 42.0 ) {
107             _h = 37;
108           }
109         }
110
111         int landa0 = _h * 6 - 183; ///< Cálculo del meridiano central
            del huso en grados.
112         double Dlanda = lonRad - (landa0 * PI_180);  ///<
            Desplazamiento del punto a calcular con respecto al meridiano
            central del huso.
113
114         if ((84 >= lati) && (lati >= 72))
115           _letter = 'X';
116         else if ((72 > lati) && (lati >= 64))
117           _letter = 'W';
118         else if ((64 > lati) && (lati >= 56))
119           _letter = 'V';
120         else if ((56 > lati) && (lati >= 48))
121           _letter = 'U';
122         else if ((48 > lati) && (lati >= 40))
123           _letter = 'T';
124         else if ((40 > lati) && (lati >= 32))
125           _letter = 'S';
126         else if ((32 > lati) && (lati >= 24))
127           _letter = 'R';
128         else if ((24 > lati) && (lati >= 16))
```

```
        else if ((24 > lati) && (lati >= 16))
          _letter = 'Q';
        else if ((16 > lati) && (lati >= 8))
          _letter = 'P';
        else if (( 8 > lati) && (lati >= 0))
          _letter = 'N';
        else if (( 0 > lati) && (lati >= -8))
          _letter = 'M';
        else if ((-8> lati) && (lati >= -16))
          _letter = 'L';
        else if ((-16 > lati) && (lati >= -24))
          _letter = 'K';
        else if ((-24 > lati) && (lati >= -32))
          _letter = 'J';
        else if ((-32 > lati) && (lati >= -40))
          _letter = 'H';
        else if ((-40 > lati) && (lati >= -48))
          _letter = 'G';
        else if ((-48 > lati) && (lati >= -56))
          _letter = 'F';
        else if ((-56 > lati) && (lati >= -64))
          _letter = 'E';
        else if ((-64 > lati) && (lati >= -72))
          _letter = 'D';
        else if ((-72 > lati) && (lati >= -80))
          _letter = 'C';
        else
          _letter = 'Z'; // This is here as an error flag to show
                         // that the latitude is outside the
                         // UTM limits

        /*!
        * Ecuaciones de Coticchia-Surace para el paso de Geográficas
        a UTM (Problema directo);
        */

        /// Cálculo de Parámetros.
        double coslatRad = cos(latRad);
        double coslatRad2 = sq(coslatRad);

        double A = coslatRad * sin(Dlanda);
        double xi = 0.5 * log((1 + A) / (1 - A));
        double n = atan(tan(latRad) / cos(Dlanda)) - latRad;
        double v = (c / sqrt(1 + e2 * coslatRad2)) * 0.9996;
        double z = e2_2 * sq(xi) * coslatRad2;
        double A1 = sin(2 * latRad);
        double A2 = A1 * coslatRad2;
        double J2 = latRad + (A1 / 2.0);
```

```
174         double J4 = (3.0 * J2 + A2) / 4.0;
175         double J6 = (5.0 * J4 + A2 * coslatRad2) / 3.0;
176         double Bfi = 0.9996 * c * (latRad - alf * J2 + bet * J4 - gam
    * J6);
177
178         /*!
179         * Cálculo final de coordenadas UTM
180         */
181         /*
182         Serial.println (" Las coordenadas GPS que se van a
    transformar son: ");
183         Serial.print (" Latitud: "); Serial.println (lati,6);
184         Serial.print (" Longitud: "); Serial.println (longi,6);
185
186         Serial.println (" Coordenadas UTM actuales: ");
187         Serial.print("UTM: "); Serial.print(_h); Serial.print("
    ");Serial.println(_letter);
188         */
189         _x = round(xi * v * (1 + (z / 3.0)) + 500000);
190         /*!< 500.000 es el retranqueo que se realiza en cada huso
    sobre el origen de
191         coordenadas en el eje X con el objeto de que no existan
    coordenadas negativas. */
192
193         _y = round(n * v * (1 + z) + Bfi);
194         if (lati < 0.0) _y += 10000000;
195         /*!< En el caso de latitudes al sur del ecuador, se sumará al
    valor de Y 10.000.000
196         para evitar coordenadas negativas. */
197         /*
198         Serial.print (" X = "); Serial.print (_x); Serial.println ("
    (m)");
199         Serial.print (" Y = "); Serial.print (_y); Serial.println ("
    (m)");
200         */
201       };
202
203       int zone(){
204         return _h;
205       };
206
207       char band(){
208         return _letter;
209       };
210
211       long X(){
212         return _x;
213       }:
```

```
214
215         long Y(){
216           return _y;
217         };
218     };
219
220     #endif
221
```